02 | 上手：一个清单应用帮你入门Vue.js

2021-10-20 大圣

《玩转Vue 3全家桶》 课程介绍 >

讲述：大圣

时长 16:27 大小 15.07M

你好，我是大圣，欢迎进入课程导读篇的第二讲。

在上一讲中，我带你了解了前端框架的基本发展历史。在为什么要学 Vue 框架这个问题上，相信你现在已经有了自己的答案。那么从今天开始，我们正式进入上手学 Vue 的阶段。

我们的专栏课程会通过故事的形式展开。故事的主角小圣是一名刚入行的前端工程师，在校期间学了点 HTML、CSS 和 JavaScript，但是不太懂框架。我是他的经理，会手把手教他在 Vue.js 这个框架里打怪升级。

小圣在学习 Vue 的过程中碰到的各种问题，同样也可能是你会碰到的问题。所以，在我带着你一起解决小圣面临的问题的同时，你的很多问题也会迎刃而解。

今天是小圣第一天入职，他只知道团队的项目是用 Vue.js 开发的，但并不熟悉 Vue 的具体技术细节，所以我决定带他先做一个清单应用，先在整体上熟悉这个框架。

如果你已经很熟悉 Vue 开发了，可以直接粗略地把本讲过一遍，直奔下一讲。在那里，我会带你梳理 Vue 3 的新特性，相信这些新特性会让你对 Vue 3 产生新的期待。

环境准备

小圣领完电脑后，首先要做的是安装编辑器和浏览器，这个不用多介绍，你也能轻松地搞定。我推荐给小圣写代码的编辑器是 VS Code，调试页面的浏览器是 Chrome。有了 VS Code 和 Chrome，我们就可以开始后面的工作了。

任务分解

如下图所示，小圣要上手开发的应用大概长这个样子：它有一个输入框，供我们输入数据；下方有一个列表，显示着所有我们要做的事情。

在输入框输入内容后，敲下回车，下面就会新增一条数据。对于每个要做的事情，你还可以用复选框标记，标记后文字就会变灰，并带有一个删除的效果，表示这件事情已经做完了。

清单应用虽然看起来简单，不过麻雀虽小，五脏俱全。其实不管入门哪个框架，你都可以写一个清单，来上手体验一下。

不过，由于小圣只有简单的 jQuery 开发经验，他在学习 Vue 的时候，首先要做的就是思想的转变。为什么要这么说呢？下面我们来对比看看 jQuery 的开发思路和 Vue.js 的开发思路有什么不同，看完你就会明白，我为什么说小圣在学习 Vue 时，首先要做的是转变思路。

比如，我们想做一个输入框，里面输入的任何数据都会在页面上同步显示。

对于这样一个前端的功能，jQuery 开发的思路是：

1. 先找到输入框，给输入框绑定输入事件；
2. 输入的同时，我们获取输入的值；
3. 再找到对应的 html 标签，更新标签的内容。

对应代码大概是这样的：

复制代码

```
<div>
    <h2 id="app"></h2>
    <input type="text" id="todo-input">
</div>
<script src="jquery.min.js"></script>
<script>
    // 找到输入框，监听输入
    $('#todo-input').on('input',function(){
        let val = $(this).val() // 获取值
        $('#app').html(val) // 找到标签，修改内容
    })
</script>
```

在实现我们想要的输入框的功能时，上述 jQuery 代码需要先找到输入框，然后持续监听输入，之后一直等待到输入值被获取，最后找到标签所在的前端页面位置，进行内容的修改。

上述的 jQuery 代码，其实是 jQuery 时代的开发逻辑的一个缩影。**而 jQuery 时代的开发逻辑，就是我们先要找到目标元素，然后再进行对应的修改。**

学习 Vue.js，首先就要进行思想的升级，也就是说，**不要再思考页面的元素怎么操作，而是要思考数据是怎么变化的。**这就意味着，我们只需要操作数据，至于数据和页面的同步问题，Vue 会帮我们处理。实际上，Vue 让前端开发者能够专注数据本身的操作，而数据和页面的同步问题，则交由 Vue 来负责。这种机制正是 Vue 当初受到开发者青睐的一个重要原因。

对于同样的输入框需求，Vue 的开发思路是：我们需要一个数据，在输入框的代码和 h2 标签的代码内使用。我们只需要操作数据，然后交给 Vue 去管理数据和页面的同步就可以了。

在 Vue 框架下，如果你想要页面显示一个数据，就要先在代码的 data 里声明数据；在输入框的代码里，使用 v-model 来标记输入框和数据的同步；在 HTML 模板里，使用两个花括号标记，来显示数据，例如{{title}}。对应代码大概是这个样子：

复制代码

```
<div id="app">
  <h2>{{title}}</h2>
  <input type="text" v-model="title">
</div>

<script src="https://unpkg.com/vue@next"></script>
<script>
const App = {
  data() {
    return {
      title: "" // 定义一个数据
    }
  }
}
// 启动应用
Vue.createApp(App).mount('#app')
</script>
```

从这个例子中，你就可以看到 Vue 在开发思路上和 jQuery 的不同。而我们要做的，就是逐渐习惯 Vue 的这种开发模式。

清单页面的渲染

在前端页面，我们在输入框输入数据，然后输入框下方要有一个列表，显示我们所有输入的值。按照 Vue 的思考方式，如果我们想实现这个功能，那么我们需要一个数组，然后使用 v-for 这个语法来循环渲染。

先看代码：

复制代码

```
<div id="app">
  <h2>{{title}}</h2>
  <input type="text" v-model="title">
  <ul>
    <li v-for="todo in todos">{{todo}}</li>
  </ul>
</div>

<script src="https://unpkg.com/vue@next"></script>
<script>
const App = {
  data() {
    return {
      title: "", // 定义一个数据
      todos:['吃饭','睡觉']
    }
  }
}
// 启动应用
Vue.createApp(App).mount('#app')
</script>
```

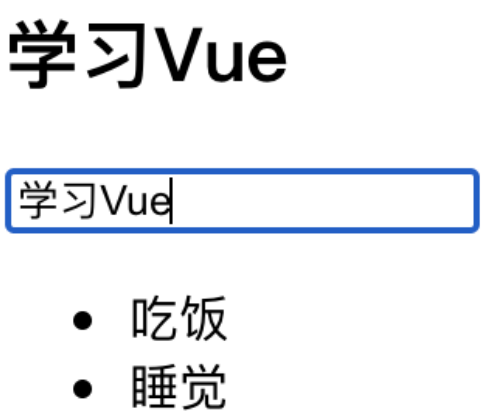
看上述代码，在 data 中，我们再定义一个数据 todos，输入一个数组。为了方便调试，我们先放两个假数据，如果我们在标签里直接写{{todos}}，就会看到显示的是一个数组，但这个不是我们想要的，我们需要的是显示一个列表。

在 Vue 中，只要是渲染列表，我们都是用 v-for 这个语法，而具体到上述代码对 v-for 语法的使用，也即：

复制代码

```
<li v-for="todo in todos">{{todo}}</li>
```

上面这行单独抽出来的代码的意思就是：我们循环遍历 todos 这个数据， 每一条遍历的结果叫 todo，然后把这个数据渲染出来，这样页面就能显示一个列表了。



处理用户交互

在上一步中，我们主要考虑的是：实现前端页面的一个输入框，以及能显示输入值的一个列表的功能。下一步，就是让用户敲回车的时候，能够让列表新增一条。采用 Vue 的思维，我们需要完成以下这几个步骤：

1. 监听用户的输入。在监听中，如果判断到用户的输入是回车的时候，那就执行一个函数。
2. 在执行的这个函数内部把 title 追加到 todos 最后面的位置，并且清空 title。

那么 Vue 如何实现这一功能呢？我们先看实现这一功能后的完整代码：

复制代码

```
<div id="app">
  <input type="text" v-model="title" @keydown.enter="addTodo">
  <ul>
    <li v-for="todo in todos">{{todo}}</li>
  </ul>
</div>

<script src="https://unpkg.com/vue@next"></script>
<script>

```

```
const App = {
  data() {
    return {
      title: "", // 定义一个数据
      todos:['吃饭','睡觉']
    }
  },
  methods:{
    addTodo(){
      this.todos.push(this.title)
      this.title = ""
    }
  }
}
// 启动应用
Vue.createApp(App).mount('#app')
</script>
```

对照上述代码，我们来看一下在 Vue 中，监听用户交互的方法。在 Vue 中，我们使用 @ 来标记用户的交互，@click 是点击，@keydown 是键盘敲下，所以就像上述代码展示的那样，如果只监听回车键，那么我们就用 @keydown.enter=“addTodo” 。

监听到用户的输入后，对于要执行的函数，我们新增一个 methods 配置。在函数内部，我们可以在 this 上直接读到 data 里的的数据，所以我们不需要考虑怎么找到标签，只需要进行如下这行潇洒的代码，就能让列表自动新增了一条， 这就是数据驱动页面的魅力。

复制代码

```
this.todos.push(this.title)
```

额外信息的显示

好了，我们现在既实现了一个输入框，以及输入数据后能够新增一条数据的列表的功能，也实现了用户在输入后的交互功能。

下一步，我们想实现标记清单中某一项是否完成的功能。但这却难住了小圣同学，因为从目前的代码设计上来看，我们的输入只能是字符串格式的内容。而我们想要实现的标记功能，却是把列表中的某一项，用灰色的字体背景和中划线来标记，以此表示这一条内容是已经完成的内容。

如果我们想实现这个功能，就需要对数据结构进行一下改造，把内容的数据类型，从简单的字符串类型改为对象。

那么数据结构要怎么改造呢？我们先直接看改造数据结构后的完整代码：

复制代码

```
  <ul>
    <li v-for="todo in todos">
      <input type="checkbox" v-model="todo.done">
      <span :class="{done:todo.done}"> {{todo.title}}</span>
    </li>
  </ul>

<script>
const App = {
  data() {
    return {
      title: "", // 定义一个数据
      todos:[
        {title:'吃饭',done:false},
        {title:'睡觉',done:true}
      ]
    }
  },
  methods:{
    addTodo(){
      this.todos.push({
        title:this.title,
        done:false
      })
      this.title = ""
    }
  }
}
</script>



<style>
  .done{
    color:gray;
    text-decoration: line-through;
  }
</style>
```

结合代码，我给你理理整个的改造思路。首先，对于 todos 数组，除了 title，还要加上一个 done 字段，来标记列表中的某一项内容是否完成，并且渲染的时候使用 todo.title。

在前面的步骤中，对于列表中每一项，我们是用无序列来表示的。但如果我们想要在列表中实现对某些选项的同时多选，那么就需要用到复选框。对于每条信息，我们都要加一个复选框，所以我们依然使用 v-model 来绑定这个 done 字段，从而实现数据里能记录用户操作的状态。

我们还需要根据 done 字段来显示某一行的样式。在 Vue 中，冒号":" 开头的属性是用来传递数据的，这里的写法的意思就是根据 todo.done 来决定是否有 done 这个 class。最后，当加上".done"的样式后，下面的左图就是我们想要的效果，而下面的右图则是涉及到".done"的相关代码：

```
<html>
  <head></head>
▼ <body> == $0
  ▼ <div id="app" data-v-app>
      <input type="text">
    ▼ <ul>
      ▼ <li>
          ::marker
          <input type="checkbox">
          <span class>吃饭</span>
        </li>
      ▼ <li>
          ::marker
          <input type="checkbox">
          <span class="done">睡觉</span>
        </li>
      ▼ <li>
          ::marker
          <input type="checkbox">
          <span class="done">学习Vue</span>
        </li>
```

进一步优化

完成前面的步骤以后，现在看起来一个清单应用最基本的功能模块、用户交互、复选框功能都已经实现了。但是为了进一步提升交互，小圣还想要增加两个功能，第一个功能是：在前端页面显示的列表的最下面，显示一下列表项目中没完成的项目的比例；第二个功能是：新增一个清理的按钮，用来删掉已经标记为完成的列表中的一条或多条数据。

那么，对于要增加的第一个功能，也即如何实现在前端页面的列表的最下方，显示一下列表项目中没有完成的项目的比例呢？小圣按照学到的知识，写出了下面的代码：

复制代码

```
<div>
  {{todos.filter(v=>!v.done).length}}
  /
  {{todos.length}}
</div>
```

把这段代码增加到上一步最后的完整代码中，运行代码，从下图所示的前端页面运行时状态中，我们能看到，其中显示的未完成比例的数据也没问题。

不过，从上述代码实现的方式上看，代码看起来很丑且性能不好，而且需要二次计算的数据，这在我们开发的需求中很常见。此外，在模板里面写 JS，看起来代码也很乱。Vue 针对这种情况，设计了一个功能，也就是**计算属性**。

我们看一下采用 Vue 的计算属性实现的，能够支持二次计算的上述功能的实现代码：

复制代码

```
<div>
  {{active}}  / {{all}}
</div>

<script>
  computed:{
    active(){
      return this.todos.filter(v=>!v.done).length
    },
    all(){
      return this.todos.length
    }
```

```
  }
</script>
```

从上面的代码中能看到，和之前采用往模板里写 JS 的办法相比，我们新增了一个属性 computed。computed 属性的配置，也即 active 和 all，都是函数。这两个函数返回的计算后的值，在模板里可以直接当做数据来用，这样把 JavaScript 的计算逻辑依然放在了 JavaScript 里，避免了过于臃肿的模板。

而且 computed 计算属性还内置了缓存功能，如果依赖数据没变化，多次使用计算属性会直接返回缓存结果，同我们直接写在模板里相比，性能也有了提升。

计算属性不仅可以用来返回数据，有些时候我们也需要修改计算属性，比如我让小圣新增一个全选的复选框，要求如下：

1. 全选框在勾选与取消勾选两个状态之间的切换，会把所有清单内的数据都同步勾选。
2. 清单内的项目如果全部选中或者取消，也会修改全选框的状态。

对于新增全选框的功能，需要满足上面的两个要求，所以全选框这个计算属性就有了修改的需求。这时候 computed 的配置就不能是函数了，要变成一个对象，分别实现 get 和 set 函数，get 就是之前的返回值，set 就是修改计算属性要执行的函数。

我们来看一下 computed 修改后的代码：

复制代码

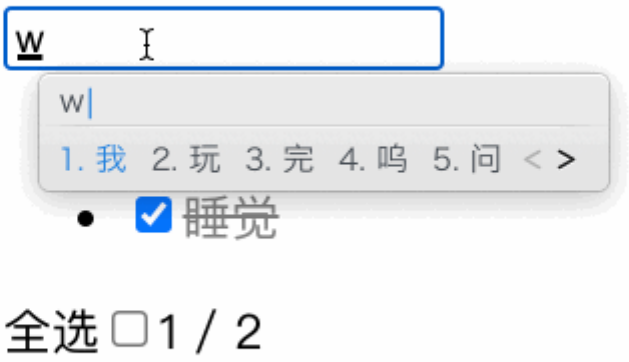
```
<div>
  全选<input type="checkbox" v-model="allDone">
  <span> {{active}}  / {{all}} </span>
</div>

<script>
computed:{
  active(){
    return this.todos.filter(v=>!v.done).length
  },
  all(){
    return this.todos.length
  },
  allDone: {
      get: function () {
        return this.active === 0
```

```
      },
      set: function (val) {
        this.todos.forEach(todo=>{
          todo.done = val
        });
      }
    }
  }
</script>
```

和没有全选框时的 computed 属性的配置代码相比，上面的代码新增了一个 allDone 的计算属性，页面中直接使用 checbox 绑定。在 allDone 的 get 函数里，对于 allDone 会返回什么值，我们只需要判断计算属性 active 是不是 0 就可以。

而 set 函数做到的就是，我们在修改 allDone，也就是前端页面切换全选框的时候，直接遍历 todos，把里面的 done 字段直接和 allDone 同步即可。

实现新增一个全选的复选框后的效果是什么样呢？我们一起来看一下：



条件渲染

在上面一部分，我们增加了在前端页面的底部显示未完成比例，和增加全选框这两个功能。除此之外，我们还需要新增一个“清理”的按钮，点击之后把已完成的数据删除，功能需求很简单，但是有一个额外的要求，就是列表中没有标记为完成的某一项列表数据时，这个按钮是不显示的。

这种在特定条件下才显示，或者隐藏的需求也很常见，我们称之为条件渲染。在 Vue 中，我们使用 v-if 来实现条件渲染。

老规矩，我们还是先看代码：

复制代码

```
<button v-if="active<all" @click="clear">清理</button>
<script>
  methods:{
    clear(){
      this.todos = this.todos.filter(v=>!v.done)
    }
  }
</script>
```

通过上述代码，我们实现了增加一个清理按钮的功能。当 active 小于 all 的时候，我们显示清理按钮，也就是说，v-if 后面的值是 true 的时候，显示清理按钮，false 的时候不显示。“@”符号的作用，我们在前面讲到监听用户交互时，已经拿 @keydown 为例说明过了，这里代码中的 @click 的作用是绑定点击事件。

我们还可以用 v-else 配合 v-if，当 todos 是空的时候，显示一条“暂无数据”的信息，具体的实现代码如下：

复制代码

```
  <ul v-if="todos.length">
    <li v-for="todo in todos">
      <input type="checkbox" v-model="todo.done">
      <span :class="{done:todo.done}"> {{todo.title}}</span>
    </li>
  </ul>
  <div v-else>
    暂无数据
  </div>
```

当我们实现了清理按钮的功能，并且也实现了列表为空时，能够显示“暂无数据”的信息后，我们看下清单应用的最终效果。

这个需求并没有考虑美观性，小圣没写太多 CSS，主要专注在 JS 的交互逻辑上。小圣这个需求做完，晚上下班的时候跟我分享了一下学习 Vue 的心得，你也可以在评论区分享一下你对 Vue 的开发的心得，我们一起交流。

总结

我们来总结一下小圣今天都学到了什么吧。入职第一天，小圣首先扭转了之前使用 jQuery 时的开发思路，并且弄明白了 jQuery 和 Vue 开发思路的区别。从寻找 DOM 到数据驱动，这是前端开发的一次巨大的变革，也是小圣同学的第一个挑战。

其次就是对 Vue 的入门使用，我带你回顾一下今天做的这个清单应用：对于这个应用，首先我们要有输入框能输入文本，并且在输入框下方循环显示清单，我们用到了 v-model，v-for 这些功能。这些 v- 开头的属性都是 Vue 自带写法，我们通过{{}}包裹的形式显示数据。

然后我们想在用户输入完成后敲击回车新增一条数据，这就用到 @开头的几个属性，@keyup 和 @click 都是绑定对应的交互事件。最后，通过 computed，我们能对页面数据的显示进行优化。我们所需要关心的，就是数据的变化，这种思维方式会贯穿小圣的整个打怪升级之路。

思考题

下班前我给小圣布置一个作业，现在所有的操作状态一刷新就都没了，这个问题怎么解决呢？

欢迎在评论区一起讨论，也欢迎你把这篇文章分享给其他人，我们下一讲见。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 45 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 01 | 宏观视角：从前端框架发展史聊聊为什么要学Vue 3？

下一篇 03 | 新特性：初探Vue 3新特性

精选留言 (42)

写留言

ch3cknull

2021-10-20

交作业：
仅前端缓存：在unmount组件时，将 组件的 data 存到 localStorage ，mount组件时取出。

如果后端有接口的话，可以在unmount时，同步到后端，挂载时请求接口...
展开

作者回复: 回答正确！

2 27

II
2021-10-20

思考题：这个涉及到页面状态保存，方法有很多，大概两类：
1. 本地储存：
a. localstorage
b. workers
c. router，也可以存到 route 中...
展开

作者回复: 赞呀，方案比我想的还多 localstorage和接口是最常见的

1 10

www
2021-10-20

全选按钮使用 set 和 get 进行处理，真是妙啊。
这一个方法就值了了

作者回复: 哈哈 感谢感谢 computed的妙用

1 6

付帅帅
2021-10-20

目前能想到的持久化方案： 一个是借助后端让数据入库，还有就是 localStorage 这种浏览器本地持久化
展开

作者回复: 赞，localStorage对我们这个Demo就够用了

4

2021-10-20

两种方式 客户端持久化 或者 服务端持久化 .

客户端：localStorage sessionStorage web sql cookie indexedDB
服务端：提供接口 客户端提交数据请求.

展开

作者回复: 回答正确！

3

桃子-夏勇杰
2021-10-22

{{active}} / {{all}} 这里为什么要做成2个属性而不是直接做个1个属性呢？

作者回复: 其实完整的应用，all还会在别的地方用到 就没有合并

2

南山
2021-10-20

打卡
1. 保存时机，在unmount的生命周期进行保存
2. 使用浏览器客户端存储，如sessionStorage，LocalStorage等api保存，保存时添加时间戳用于比对新老数据，使用服务端保存接口，将数据持久化到数据库，考虑接口请求失败时重试机制和友好提示...

展开

作者回复: 赞赞赞，还考虑了保存失败的异常

2

Geek_be2349
2021-10-20

使用 localStorage 保存数据

展开

作者回复: 回答正确

2

就不吃苹果

2021-10-20

将数据存储到storage里面，在created的时候从浏览器本地获取数据吗？

作者回复: 回答正确

2

杨村长

2021-10-22

交作业：将数据存入localStorage，刷新时再取出来展现
1.保存：watch一下todoList，变化存入
2.读取：data设置为localStorage中读出的数据

作者回复: 100分100分

1

木咀

2021-10-21

尝试着用组合式API，在<script setup> 声明
state=reactive({对象})；
const a,b,c = toRefs( );
发现Script代码里面，通过 toRefs返回的，必须用 a.value来赋值，才可以保留响应式效果；...

展开

作者回复: 复杂项目和长期维护的项目，类型系统还是必备的

1

肆水流歌
2021-10-21

看了大家的留言，我发现我只会localstorage，是我太菜了

作者回复: 哈哈 其实localStorage就已经足够了

1

Harry
2021-10-21

老师晚上好！
`Vue.createApp(App).mount('#app')`
这一句总是会提示一个错误：
`Uncaught TypeError: Vue.createApp is not a function`
展开

作者回复: 你好，可以先看下代码，需要先通过一个script标签引入Vue

1 1

By:小新
2021-10-20

首先采用vuex进行app运行时数据状态保存
然后再卸载组件之前拦截，将vuex的状态持久化到
localStorage
展开

作者回复: 赞赞赞

1

3.141516
2021-10-20

之前学过小程序，发现微信小程序应该就是借鉴 vue 的风格，数据驱动、for、if 等等

作者回复: 小程序其实借鉴了Vue和React的混合风格，模板的for,if像vue，代码里的setData像React

1

Alias

2021-10-20

“jquery -> vue/react: 不要再思考页面的元素怎么操作，而是要思考数据是怎么变化的。这就意味着，我们只需要操作数据，至于数据和页面的同步问题，框架会帮我们处理”

作者回复: 赞

1

cwang

2021-10-20

学习任何框架，都可以以这个To do List为起始点。

展开

作者回复: 是这样的，包含了常见的交互和数据逻辑

1

吴晨辉

2021-10-20

课后作业：
本人背景：纯后端来学前端
思路：将数据缓存在浏览器页面刷新后不清除的地方（可以是后端，也可以是浏览器本地储存）
如果是后端，那么需要绑定一个后端接口，刷新页面重新获取数据...

展开

作者回复: 赞赞赞，还考虑了异常情况

1

cx

2021-10-20

监听todos的数据，然后通过缓存的方式解决

展开

作者回复: 你好，那么缓存到哪里呢

1

一纸信笺

2021-10-20

最先想到解决问题的办法是利用缓存将当前数据进行存储

作者回复: 思路是正确的，就看用哪个缓存了

1